GFP-X: A Parallel Approach To Massive Graph Comparison Using Spark

Stephen Bonner, John Brennan, Georgios Theodoropoulos, Ibad Kureshi, Andrew Stephen McGough
School of Engineering and Computing Sciences
Durham University, Durham, UK
{*s.a.r.bonner, j.d.brennan, georgios.theodoropoulos, ibad.kureshi, stephen.mcgough*}*@durham.ac.uk*

***Abstract*—The problem of how to compare empirical graphs is an area of great interest within the field of network science. The ability to accurately but efficiently compare graphs has a significant impact in such areas as temporal graph evolution, anomaly detection and protein comparison. The comparison problem is compounded when working with massive graphs containing millions of vertices and edges.**

This paper introduces a parallel feature extraction based approach for the efficient comparison of large unlabelled graph datasets using Apache Spark. The approach acts by producing a 'Graph Fingerprint' which represents both vertex level and global level topological features from a graph. By using Spark we are able to efficiently compare graphs considered unmanageably large to other approaches. The runtime of the approach is shown to scale sub-linearly with the size and complexity of the graphs being fingerprinted. Importantly, the approach is shown to not only be comparable to existing approaches, but on when comparing topology and size, more sensitive at detecting variation between graphs.

***Keywords*-Graph Similarity; Feature Extraction; Spark;**

I. Introduction

Network science is an interdisciplinary field for the study of detailed real-world phenomena by viewing them as a graph. In many domains, being able to compute the similarity between two graphs is extremely valuable. Such domains include: anomaly detection [1] [2], protein comparisons [3] [4] and the study of temporal graph evolution / link predication [5]. Thus, graph comparison and specifically similarity measurement is an area of increasing research interest.

The terms graph and network are often used interchangeably within the literature, however, in this work we shall use the term graph without loss of generality. We define a graph $G = (V, E)$ as a finite set of vertices (sometimes referred to as nodes) – V – and a finite set of edges – E. The elements of E are an unordered tuple $\{u, v\}$ of vertices $\{u, v\} \in V$. It is possible for a graph to have a set of labels associated with vertices, edges or both. In such cases we can define a graph $G = (V, E, L)$, where L is a set of labels. A label contains additional information about an edge, vertex or the graph itself, for example a person's name or age within a social network.

There are many definitions of similarity between graphs [6] [7] [8], however, they can be split into two categories – those which can only be applied to labelled graphs and those which can to applied to graphs irrespective of labelling. When labels are available, similarity can be based on such metrics as the number or similarity of labels appearing in both graphs, however, when labels are not present similarity is based on topology comparison. In this paper we focus on topology comparison of unlabelled graphs.

A number of considerations need to be addressed when computing the topological similarity between graphs to ensure accurate comparison. For example, two graphs might appear very similar when considering the individual edges between vertices, yet be of completely different graph sizes. A counter example being two graphs which are of comparable size, yet have vastly different degree distributions (the distribution of edges between the vertices within a graph).

Most importantly, any comparison approach should be able to scale to the so-called high volume (massive) graphs (vertices and edges) seen in such areas as social media. Graph processing techniques are being applied in a broader range of data driven fields, where data volumes are large and constantly increasing, resulting in more graphs of larger sizes [9]. The current Facebook social network, for example, is said to contain over one billion vertices and is still growing [10]. This dramatic increase in the quantity of data means that ever larger graphs need comparing against one another. This has a significant effect upon graph similarity measures, as any such algorithm needs to be accurate, computationally efficient and needs computing in realistic time-scales – requiring the use of parallel techniques.

In this paper we present a new parallel approach for extracting Graph Fingerprints based on our initial work [11], a compact but representative abstraction of a graph, with numerous potential applications within field such as machine learning. The new approach, entitled Graph Fingerprint Extract (GFP-X), utilises Apache Spark and GraphX to massively decrease feature extraction times whilst increasing the maximum size of processable datasets. We demonstrate an application of the fingerprint approach for the comparison of graphs, named Graph FingerPrint Comparison (GFP-C), that is label independent as it exploits only the topology of a given graph in order to compare similarity.

The key contributions of this paper are:

1) **Scalability** - We present a distributed approach using Apache Spark and GraphX to measure graph similarity – the first approach to explore the use of these

systems. The approach is shown to scale sub-linearly to increases in dataset size and to be effective when processing graphs of over 100 million vertices, an order of magnitude greater then seen in the literature. The approach also scales from running on a single machine to a dedicated cluster.

2) **Sensitivity** - We demonstrate that the GFP-C approach is more sensitive at detecting variations in graph size and topology than existing approaches. This is achieved by the exploiting the combination of both global and local features when performing graph comparisons.
3) **Reproducibility** - The entire codebase has been open-sourced, along with the scripts to run the presented experimentation on the Stanford Network Analysis Project (SNAP) public datasets.

In Section II we discuss related work, motivation is presented in Section III, Section IV highlights the fingerprint generation method, Section V details the comparison of fingerprints, Section VI details the GraphX implementation of both GFP-X and GFP-C, Section VII presents empirical results and Section VIII draws conclusions and explores possible future research.

II. Previous Work

It has been argued [6] [7], that the various label dependant and independent methods for graph comparisons can be further categorised into three major cross cutting classes: graph-isomorphism based methods, iterative methods and feature extraction based methods. Prior work [6] [11] has shown feature extraction based methods to be more scalable and flexible, thus are the focus of this paper. Readers are referred to past reviews for more information about *Graph-Isomorphism* and *Iterative Method* based graph comparison methods [6] [11] [7].

A. Feature Extraction

A range of features are extracted from a graph for comparison with other graphs the more similar two graphs the more similar their features. Feature extraction based methods have advantages over other approaches as they can be highly scalable – thus have faster runtimes [7]. However, determining which features to extract to give the best, yet most compact, representation of a graph, is an area of active research [8].

One such feature extraction method presented by Roy et al. extracts a variety of centrality measures (used to rank the importance of a vertex within a graph [12]) and uses them for graph comparison [13]. This approach requires that the graphs are labelled and has not been validated on anything but small graphs, with the largest dataset only having 20,000 vertices. An alternative feature extraction method presented by Papadimitriou et al. has been explored to measure the similarity between snapshots of a graph of links between webpages [8]. In this approach several similarity measures are tested on time-series of graph data with the goal of detecting anomalies between time-steps. However, many of the methods tested rely on labeled data to compute similarity.

The NetSimile algorithm [6] relies upon extracting details about the 'EgoNet' (A vertex's EgoNet is every other vertex which is connected to it in its local neighbourhood) for each vertex within a graph which is then compared, via a distance metric, with results from other graphs. In the presented results, NetSimile is shown to be independent of graph size when making the comparison and only considers the similarity of the underlying linking model, meaning that two graphs of vastly different scales could be identified as 'similar'. NetSimile does not run on a parallel graph analytic platform, thus limiting the size of graph it can compare.

Feature extraction has been explored outside of similarity measurement as a way of classifying graphs based on comparisons between global features and labels [14]. Additionally feature extraction has been explored by the anomaly detection community as a way of detecting unusual elements or events within static and temporal graphs [1].

B. Parallel Graph Similarity Measures

To date, there has been little work on comparing graphs in parallel or on how to efficiently compare graphs of millions of vertices or edges. One such recent approach is entitled 'DeltaCon' [15] which is an approach to compare the similarity of two graphs based upon common labelled vertices. Whilst the approach is stated to be scalable, only a dataset of 16M vertices is tested and a parallel version is only hypothesised, not implemented. A parallel approach for graph similarity using a Message Passing Interface (MPI) compute cluster has been created [16]. The approach is shown to scale to over 1000 compute cores and to a graph size of over 1M vertices. However the approach does not produce a final similarity score for two graphs, instead the algorithm matches the similarity of each vertex in one graph to every vertex in the second, thus is very computational expensive and cannot scale to truly massive graphs.

III. Graph Comparisons

A. Background

The research behind the GFP-X and GFP-C approaches is part of a larger body of work investigating new machine learning techniques to study and predict the temporal evolution of massive graphs. An accurate way of comparing an empirical graph to a synthetically generated graph, as predicated by the machine learning algorithm, is needed. Any difference between the empirical and generated datasets can be used to validate the generation method. Thus accuracy and sensitivity (to small changes in graph structure) are crucial. The development of the GFP-C approach was required when we found existing serial methods for graph comparison were unable to scale to massive internet scale graphs and

slow when comparing even modest sized graphs. In this context, we define two graphs to be similar if they share similar global and micro (vertex and edge) level topological features.

Our approach is driven by the requirements:

1) **Scalability** - Highly scalable to massive graphs of millions of vertices/edges, and capable of computing the similarity in a finite time.
2) **Sensitivity to Graph Size** - Taking the size and order of the graphs into consideration.
3) **Sensitivity to Similar Topologies** - Detecting the difference between graphs which are highly structurally and topologically similar.
4) **Label Free** - Able to perform comparisons without requiring labeled datasets, although the approach should still function when they are available.
5) **Low Number of User Defined Parameters** - A minimum number of user defined parameters should be required to measure graph similarity.

B. Theoretical Approach Overview

Our approach is comprised of two distinct stages: The generation of a graph's fingerprint (GFP-X) and the comparison of these fingerprints (GFP-C). The GFP-X approach takes the high dimensionality inherent in complex graphs and reduces this down into two fixed length feature vectors. The GFP-X approach achieves this by extracting micro and macro features from the given graph, allowing it to capture both the macro and microscopic topological features. The decision to extract both vertex level and global level features was driven by the desire to make the comparison between graphs more sensitive to small variations in the underlying graph topology and the overall size of the graph than the current state of the art methods [6].

The GFP-X approach is broken down into three stages: Vertex Level Feature Extraction, Vertex Level Feature Creation and Global Level Feature Extraction. These stages are executed for each graph to generate their fingerprint which can be used to compare graphs or can be stored for later use in some other task. After the vertex features have been extracted from the graph, they are then aggregated during the vertex feature creation stage. In addition, global features are also extracted from each graph. It is worth noting that the GFP-X approach can be extended to include any vertex or global level feature, not just those detailed in this paper.

The GFP-C approach then computes the similarity between any two graphs using the following stages:

1) Vertex Level Comparison 2) Global Level Comparison 3) Final Similarity Score Generation.

Both the Vertex and Global generation produce a feature vector for each graph. Graphs can then be compared by computing the distance between their feature vectors - in this work we use the Canberra distance metric [17]. This results in two separate similarity scores, one comparing the vertex level topology and one the global level similarity. The last stage is to combine these two scores to produce the final similarity score between two graphs. In the next two sections, both the GFP-X and GFP-C approaches are described in greater depth.

IV. Generating Graph Fingerprints

A. Vertex Features

The GFP-X approach extracts a variety of features from each vertex within a graph. Although a wide selection of vertex feature metrics exist, each exhibits different characteristics in terms of topological sensitivity and runtime. Through experimentation which has been omitted here for brevity, we have determined that the seven features detailed in Table I results in the best balance between topological sensitivity and runtime. A greater exploration of the features can be found in our previous work [11]. However, other features could also be incorporated if other characteristics of a graph are important. For each of the seven vertex features detailed in Table I, a value is computed for each $v \in V$.

B. Global Features

In order to make the GFP-X approach sensitive to the global features of a given graph, a selection of global features are extracted. The global features, chosen to represent each graph, were selected due to their ability to capture key elements of global graph topology, whilst also being efficient to compute in a distributed environment. A vector is used to represent these six global features:

Graph Order - Defined as: $|V|$.

Number of Edges - Defined as: $|E|$.

Number of Triangles - The number of triangles, α, for a given graph is the number of vertices which form a triangle, with a triangle being a set of three vertices with an edge between every member.

Maximum Total Degree Value - This represents the total number of edges the most connected vertex in the graph has to other vertices.

Number of Components - This is the total number of components within the graph, with a component being a subgraph in which there is a possible path between every vertex, whilst vertices in different components have no possible path between them.

Number of Vertices In Largest Component - This is the total number of vertices within the largest component in the graph.

C. Feature Creation

The matrix, $VF_{m,n}$, where $m = |V|$, contains all the vertex feature scores as defined in Section IV-A, and $n = |F|$

Table I
GFP-X VERTEX LEVEL FEATURES

Feature Name	Label	Equation	Source
Eigenvector Centrality Value - The Eigenvector centrality is used to calculate the importance of each vertex within a graph. Formally the Eigenvector centrality can be written as an eigenvector equation, where λ is the largest eigenvalue, $\mathbf{A}$ is the graph in adjacency matrix from and $\mathbf{x}$ is the eigenvector centrality.	$\mathbf{Ax}$	$\mathbf{Ax} = \lambda\mathbf{x}$.	[18]
Two-Hop Away Neighbours - The number of two-hop away neighbours from the current vertex v gives an indication of how connected, and thus how important, a vertex's neighbourhood is within the graph, where $N(v)$ is every vertex incident on the current vertex v.	th_v	$th_v = \frac{1}{\lvert N(v)\rvert} \sum_{\forall j \in N(v)} d^+(j)$	[6]
PageRank Score - The PageRank centrality method was originally developed by Google, however it is now commonly used to measure the local influence of a vertex within a graph. Where where $\Gamma^-(v)$ is the set of incoming neighbours of v, N is the total number of vertices, $d^+(u)$ is the out-degree of u and d is a constant damping factor (0.85 for this work).	$PR(v)$	$PR(v) = \frac{1-d}{N} + d \sum_{u \in \Gamma^-(v)} \frac{PR(u)}{d^+(u)}$	[19] [20]
Average PageRank of Neighbourhood - The average PageRank of the neighbourhood is the mean of each PageRank score within a vertices neighbourhood, where $PR(j)$ is the PageRank score calculated in the previous step.	$N_{PR(v)}$	$N_{PR(v)} = \frac{1}{\lvert N(v)\rvert} \sum_{\forall j \in N(v)} PR(j)$	[11]
Total Degree - This is the sum of both the in and out degree for the vertex v.	td_v	$td_v = \Gamma^-(v) + d^+(v)$	[11]
Local Clustering Score - The local clustering score for vertex v represents the probability of two neighbours of v also being neighbours of each other, where Φ is the number of pairs of $v's$ neighbours which are themselves connected.	c_v	$c_v = \frac{2\Phi}{d^+(v)(d^+(v)-1)}$	[21]
Average Clustering of Neighbourhood - The average clustering score of the neighbourhood is taken for each vertex by taking the mean of all the local clustering scores for the vertex's neighbourhood, where c_j is the local clustering score computing in the previous feature extraction step.	nc_v	$nc_v = \frac{1}{\lvert N(v)\rvert} \sum_{\forall j \in N(v)} c_j$	[11] [6]

(F is the vector of features for each vertex):

$$VF_{m,n} = \begin{pmatrix} f_{1,2} & \cdots & f_{1,n} \\ f_{2,2} & \cdots & f_{2,n} \\ \vdots & \ddots & \vdots \\ f_{m,2} & \cdots & f_{m,n} \end{pmatrix}$$

In order to create the graph fingerprint, we need to reduce the dimensionality of the feature matrix down to a more compact vector. To perform this transformation, a series of metrics are taken for each of the feature columns in the matrix. The metrics chosen are the mean, standard deviation, variance, skewness, kurtosis, minimum value and maximum value. These are frequently used and well understood methods to capture the numerical variation within a range of values. After this has been completed, the resulting vertex feature vector $\overrightarrow{v_{g1}}$ for graph $G1$ can be created. The vertex feature vector contains the eight aggregation scores for each column in the feature matrix which are concatenated together:

$$\overrightarrow{v_{g1}} = (\bar{x}_1, \sigma_1, \sigma_1^2, Skew[x]_1, Kurt[x]_1, x(1)_1, x(n)_1, ... \\ , \bar{x}_n, \sigma_n, \sigma_n^2, Skew[x]_n, Kurt[x]_n, x(1)_n, x(n)_n).$$

V. COMPARISON OF GRAPH FINGERPRINTS

The GFP-C approach must compare the fingerprints of two graphs to compute their similarity. After extensive experimental evaluation and similar to [6], the Canberra distance was selected to compare the numerical distance between the fingerprints. Other distance metrics tested including the Bray, Correlation, Chebyshev, Cosine and Manhattan but these were found to be insensitive when the feature vectors were highly similar, or produced unintuitive results such as a high similarity scores for highly dissimilar graphs. The Canberra distance between two vectors of n dimensions is:

$$CD(\overrightarrow{p}, \overrightarrow{q}) = \sum_{i=1}^{n} \frac{|p_i - q_i|}{|p_i| + |q_i|}.$$

The Canberra distance is able to accurately detect changes close to zero, which makes it ideal for detecting small variations between graphs which might be highly topological similar – one of the key goals for the GFP-C approach. The Canberra distance is used to compare both the distance between the vertex feature vectors and the global feature vectors. Two graphs are more 'similar' the closer the result of the Canberra distance is to zero, with a score of zero indicating that the graphs are 'fingerprint' identical.

A. Final Similarity Score Generation

The GFP-C approach returns two similarity scores, one for the distance between the vertex feature vectors vf_s and one for the distance between global vectors gf_s for the two graphs being compared. These two scores can be used independently to compare the global and local topological structure as separate entities. However, the GFP-C approach can produce a final similarity score between the two graphs, using the following aggregation - $FinalSimScore = vf_s + \gamma gv_s$. Where γ is a user controllable parameter to control the weighting of the difference between the global feature vectors in the final similarity score.

VI. GFP-X Implementation

A. Apache Spark and GraphX

Apache Spark is a general-purpose parallel computing framework for processing massive datasets [22], the core of which is the Resilient Distributed Dataset (RDD) abstraction. An RDD is a read only collection of data partitioned across a set of Spark cluster machines and cached in memory. The RDD concept has further been expanded via the higher-level DataFrames, which arrange the distributed collection of data into labelled columns similar to a traditional relational database [23].

GraphX is a system for processing of graph datasets using Spark [24]. It includes a variant of Googles Pregel API – the first of the 'Think Like A Vertex' (TLAV), designed to bring the scalability of a Map / Reduce like system to graph processing [25]. Graphs are represented as specialised versions of RDD's and thus can be parallelised across a cluster. GraphX includes a selection of primitive graph algorithms including connected components and triangle count but additional algorithms must be implemented by the end user using one of the available GraphX graph traversal API's: Pregel and Aggregate Messages.

B. Parallel Feature Extraction

Both the GFP-X and GFP-C approaches are written in Scala for the Apache Spark GraphX package. Spark was chosen due to it's ability to scale across a distributed environment and it's use of in-memory computation. As the main goal of GFP-X was scaleability, using Spark allowed this to be achieved. GraphX offers a range of implicit functions for extracting common features from a graph, such as triangle counting, PageRank and connected components – where ever possible, these methods were utilised. Any features not provided by GraphX must be implemented via one of the available graph traversal algorithms. To implementing the non-implicit features for GFP-X, the Aggregate Messages API was utilised. Previous research has shown that key statistics about a vertex neighbourhood [6] can be very powerful in it's identification. The Aggregate Messages API passes information from a vertex to all it's neighbours and can be considered conceptually as Map / Reduce for graphs [24]. To use the Aggregate Messages API, a send message and merge message function must be created to perform the desired computation. The send message function, analogous to a Map, controls what message is sent by every vertex within a graph. The merge message, analogous to a Reduce, controls the aggregation of multiple messages arriving at the same vertex to create a single result. This process is performed in parallel across the Spark cluster.

The Aggregate Messages API is used in three of the vertex features for GFP-X; the mean PageRank score, number of two hop away neighbours and mean local clustering score for a vertex's neighbourhood. To capture the mean PageRank score for a vertex's neighbourhood, the PageRank score, computed for each vertex using the implicit GraphX function, is used as the attribute to be passed in the send message as well as a counter variable. This results in each vertices PageRank score being sent to all it's neighbours. The merge message function then sums the incoming PageRank score messages at each vertex and divides by the total number of counters received, resulting in each vertex having the mean PageRank score for it's neighbourhood. The methodology is a generalised way of capturing the mean of any vertex feature across it's neighbourhood using it also for the mean neighbourhood local clustering score and number of two hop away neighbours. This method is extremely efficient and is fully parallelised across a cluster. The method could be expanded to aggregate a feature from multiple hops away from a vertex, capturing information about it's extended neighbourhood, using multiple iterations of the send-merge process.

All the features for GFP-X and their extraction method are detailed in Table II. Each vertex feature is returned as a VertexRDD, containing the vertex ID and the feature value. The global features are returned as a single DataFrame containing all global feature values. In order to scale to massive graphs, even when running on a single machine, memory management is a key concern. Spark allows data to be cached in memory to improve application performance, but programs can be unstable if the data requirements exceeds the amount of available memory. Due to this, we allowed the graph to cache to disk if memory space is limited. To improve the memory footprint of GFP-X, each feature is extracted and then immediately aggregated so that the original VertexRDD can be removed from memory.

C. Parallel Graph Comparison

The function for feature creation utilises the Spark DataFrame API, which allows for each set of vertex features to be aggregated efficiently and in parallel using the implicit statistics functionality. Once all the features have been aggregated, they are joined to create a vertex and global feature vector, both of which are stored as DataFrames. To compute the similarity between two graphs, these feature vectors are compared using the Canberra distance which

Table II
GFP-X FEATURE EXTRACTION METHOD

Feature	Extraction Method
Eigenvector Centrality Value	Extracted using the Sparkling-Graph package [26].
PageRank Score	Extracted using the implicit GraphX method.
Average PageRank of Neighbourhood	Extracted using Aggregate Messages mean neighbourhood method described in section VI-B.
Total Degree	Extracted by counting the number of vertices incident on each vertex.
Two-Hop Away Neighbours	Extracted using the Aggregate Messages methodology by each vertex sending the number of neighbours it has to it's neighbourhood.
Local Clustering Score	Extracted via the Sparkling-Graph package.
Average Clustering of Neighbourhood	Extracted using the mean neighbourhood Aggregate Messages method.
Graph Order	Extracted by counting the number of vertices within the VertexRDD.
Graph Size	Extracted by counting the number of edges within the EdgesRDD.
Number of Triangles	Extracted using the implicit GraphX function and a custom Map / Reduce function.
Number of Components	Extracted via the implicit GraphX function.
Number of Vertices In Largest Component	Extracted via a custom Map / Reduce method.

has been implemented using the RDD API. The two vectors being compared are first joined together, then a single Map / Reduce iteration can be used to compute the distance. In the Map phase, the absolute difference between each vector elements is divided by their absolute sum. These results are then summed in the Reduce phase. Using Apache Spark for all components, not just the graph feature extraction, of GFP-X and GFP-C, ensures that they will still be scaleable as graph datasets continue to grow.

The GFP-X and GFP-C frameworks have been open sourced under a GPLv3 licence and are available on GitHub[1]. In addition, the code used to run each experiment, generate the synthetic datasets used and the implementation of NetSimile, written in the Graph-Tool package [27], are also available in the same repository.

VII. RESULTS

In this section, the GFP-C approach is assessed against the criteria as discussed in section III-A. In each experiment, GFP-C is compared to the current state of the art feature extraction graph comparison method – NetSimile [6]. As both the GFP-C and NetSimile approaches generate their final similarity scores using the Canberra distance, their results are directly comparable. It is worth highlighting that other distance metrics, used in place of the Canberra distance, would produce similar disparities between the results of the two approaches. When using the Canberra distance metric to compare graph feature vectors the closer to zero the result, the more similar the graphs. A larger distance score indicates the graphs to be more topologically dissimilar.

A. Experimental Setup

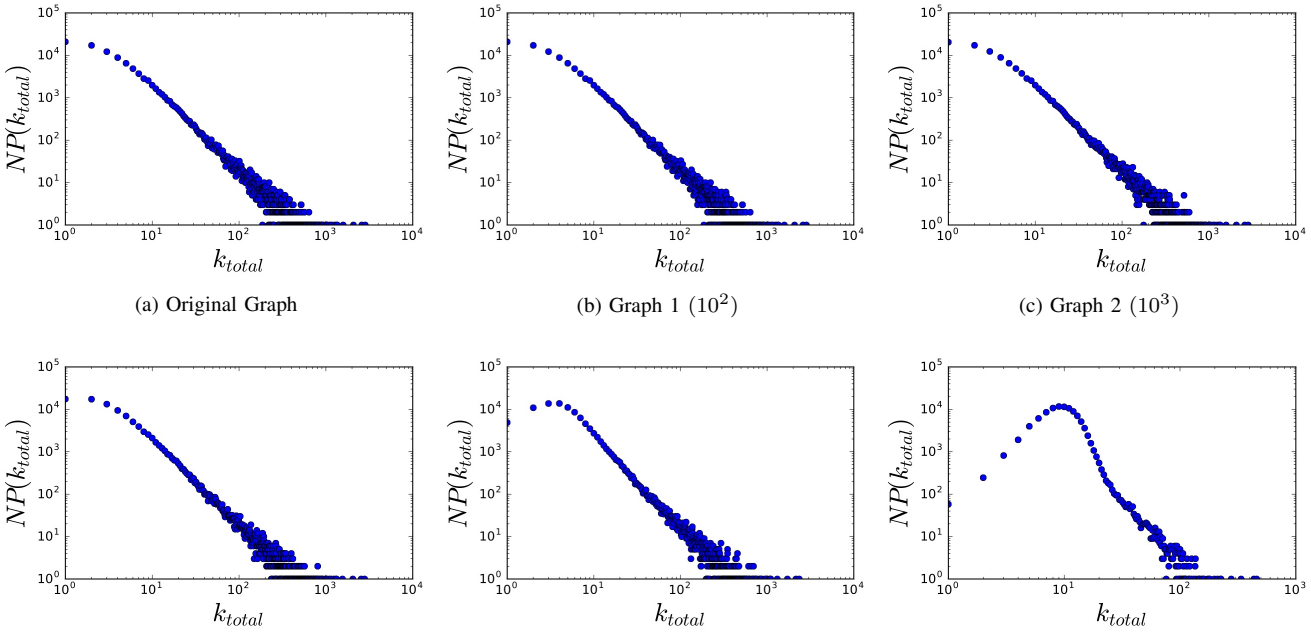
All the experiments presented in this paper were performed upon a small development Hadoop cluster comprised of a head node with a 6C Intel Xeon E5-2609v3, 64GB RAM and 1TB of SSD storage. In addition, the cluster contains 4 worker nodes each with 2 * 8C Intel Xeon E5-2630v3, 64GB RAM and 1TB of SSD storage. All nodes in the cluster are connected via a dedicated SFP+ 10Gb network and run the same software stack of CentOS 7.2, Java 1.8, Scala 2.10.5, Apache YARN 2.7.1 and Apache Spark 1.6.1. All experiments using Spark were run using YARN to allocate cluster resources in the form of containers.

[1]https://github.com/sbonner0/GFPX-GraphSimilarity

Table III
GRAPH DATASETS

Dataset	$\|V\|$	$\|E\|$	$\%VinLCC$	α
soc-Slashdot0902	82168	948464	100	602592
ca-HepPh	12008	118521	93.3	3358499
com-DBLP	317080	1049866	100	2224385
loc-Gowalla	196591	950327	100	2273138
wiki-Talk	2394385	5021410	99.8	9203519

For all experiments, γ was set to 2 to increase the weightings of the global features in the final similarly score. The synthetic graphs used throughout the results section (including Forest Fire [28] and Erdős-Rényi [29] random graphs) were generated using the SNAP 2.4 C++ graph analysis package [30]. The Forest Fire generation method was introduced by Leskovec, and produces more realistic synthetic graphs than the frequently used Barabási-Albert as it replicates more features seen in empirical graphs [28]. For all Forest Fire graphs used in the results section, the forward burning probability was set to 0.35 and the backwards burning probability set to 0.32. Please see [28] for a more detailed discussion of the forward and backward burning process. These values produce graphs which approximately follow $|E| = |V|*4$. The empirical data used was taken from the widely used Stanford Network Analysis Project (SNAP) datasets [31]. A summary of the datasets used can be seen in Table III. The datasets are from a range of domains including collaboration, communication and social networks.



(a) Original Graph (b) Graph 1 (10^2) (c) Graph 2 (10^3)

(d) Graph 3 (10^4) (e) Graph 4 (10^5) (f) Graph 5 (10^6)

Figure 1. Change In Degree Distribution After Rewiring Process.

B. Random Rewire Process

To demonstrate that the GFP-C approach is highly sensitive to the underlying topology of a given graph, the edges in a Forest Fire graph with 100,000 vertices were re-wired in a random fashion. Figure 1 shows how the degree distribution of the original graph was altered by the random rewiring process, where the number after the graph name indicates the quantity of edges rewired. The Figure plots the number of vertices $NP(total)$ with a specified total degree value k_{total} and illustrates how the internal connectivity of the original Forest Fire graph is altered as more edges are rewired. The rewire process alters a given source graph's degree distribution by randomly altering the source and target of a set number of edges according to the Erdős-Rényi random model. During this re-wire process, it is not guaranteed that the source or target of the edge will be altered, indeed it is not always possible due to the graphs topology. The rewiring process does not change the total number of edges or vertices within the graph.

C. Sensitivity to Variations in Topology

For the results presented here, an original Forest Fire graph (with 10^5 vertices) was compared to each of the rewired graphs (discussed in section VII-B to measure the similarity between them. Figure 2 shows that GFP-C is sensitive to the changes in the topology of the graph, with an increase in the percentage of the graph rewiring always being detected as more dissimilar to the source graph. The result shows that, not only is GFP-C comparable to the state of the art method NetSimile, but it is more sensitive to topological change due to the higher value of the Canberra distance.

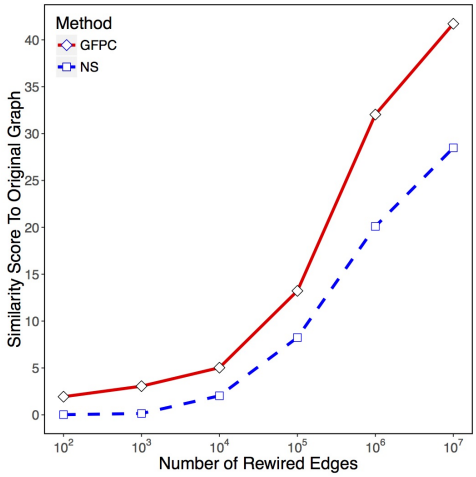


Figure 2. Sensitivity to Graph Topology

D. Sensitivity to Variations in Size

The GFP-C approach was tested for its sensitively at detecting variations in global graph size. For this experiment, a random Forest Fire graph G_o was generated with $|V| = 10^4$ and $|E| = 10^{4.6}$. To compare with the source graph, six new graphs were generated again using the Forest Fire method each with varying numbers of vertices and edges. As the Forest Fire method was used to generate all graphs, they will be highly structurally similar in their topologies. The results

comparing the GFP-C and NetSimile method for sensitivity to variations in graph size are displayed in Figure 3. In the figure, graphs of varying sizes were compared to the original graph G_o to generate the similarity score.

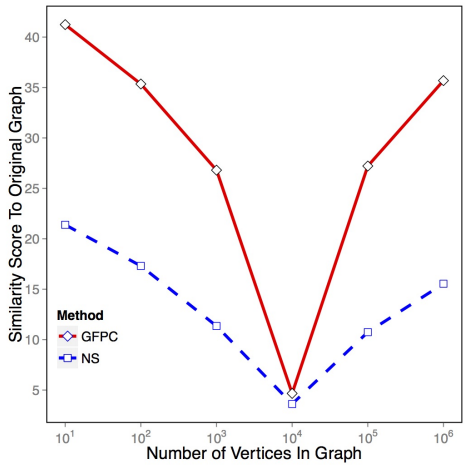


Figure 3. Sensitivity to Graph Size

Figure 3 shows that the GFP-C approach is more sensitive to variations in graph size than the NetSimile method, with a change in size of the graph always detected as more dissimilar to the source graph. It is interesting to note that GFP-C detects the graph of the same size as the source graph as being highly similar, showing that it is strongly effected by global graph size when making comparisons.

E. Runtime Analysis

The final criteria evaluated was the the runtime of the GFP-X feature extraction algorithm across a range of empirical data sources, as well as comparing it to NetSimile. This is the crucial experiment as the reason for implementing GFP-X in GraphX was to increase both runtime performance and the size of graphs that can be compared. For this comparison, we have implemented the NetSimile approach in Graph-Tool, a highly efficient C++ graph analysis library which uses OpenMP to scale across multiple cores in a shared memory system [27]. All the measures of runtime presented incorporate reading the graph data into memory from HDFS or Disk as well as the YARN scheduling and allocation decision times. As such, the presented runtimes are the total time taken to produce a final result from the initial job submission. As NetSimile is not a distributed approach, it's timings were obtained by running it upon a single node from within the cluster. For fair comparison, GFP-X was also run upon a single node in addition to the full cluster.

Figure 4 shows the runtime of the feature extraction stages for both GFP-X (Running on 1 $(1E)$ and 12 $(12E)$ Spark executors on the cluster) and NetSimile, across the datasets in Table III, with the results being the average of five experiments and the error bars being one unit of standard deviation. Whilst a direct comparison is difficult, due to GFP-X and NetSimile being implemented in different languages, the figure does highlight some interesting results. Firstly, it is clear that when running upon a single compute node GFP-X is significantly, often by over an order of magnitude, faster than the C++ based NetSimile. Secondly, due to the comparatively small size of datasets used, running across all nodes in the Spark cluster does not always result in a decrease in runtime. It's only when running on the largest dataset, *wiki-Talk*, that the inherent costs associated with distributing data across the network becomes worthwhile.

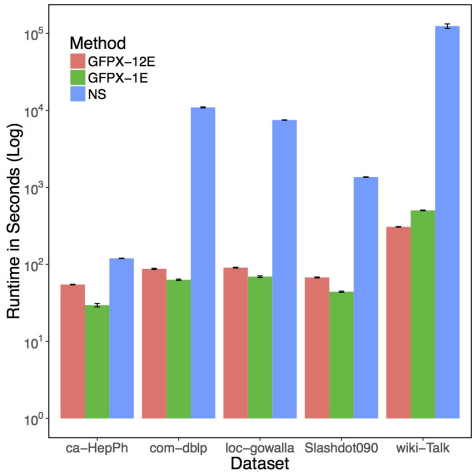


Figure 4. Runtime Performance Across Empirical Datasets

In addition to testing on empirical datasets, the run time of generating a single fingerprint using the GFP-X approach was evaluated across a range of synthetic Forest Fire and Erdős-Rényi graphs when running across the full five node Spark cluster. As the number of vertices was increased in the generated data, the number of edges was kept such that $|E| = |V| * 2$. These experiments were performed to assess the relationship between number of vertices within a range of topologically varying random graphs and the runtime of GFP-X. Again, all experiments were repeated five times and the error bars being presented as one unit of standard deviation. The runtime of Apache Spark and GraphX jobs are significantly affected by several key user configurable parameters which control how resources are allocated to the job and how many RDD partitions that the data is stored across. For a fair comparison the number of containers, cores, partitions and memory was kept constant across each dataset size. Due to this, the presented runtimes are not the lowest achievable and could have been improved with optimal parameter selection for each dataset size. However the exploration of the parameter space is out of the scope of this paper. It is also worth noting that the implicit algorithms for counting connected components in GraphX currently contains an error in the code when scaling to massive graphs, so for all the runtimes measured below this global feature

has not been extracted.

Figure 5 shows how the runtime of the GFP-X approach responds to increases in the number of vertices within a Forest-Fire graph. The additional line shows a linear relationship between dataset size and runtime. This figure shows that GFP-X responds in close to a sub-linear fashion to increases in the number of vertices within a graph. It can be seen that an increase of an order of magnitude in the number of vertices, never corresponds with same increase in runtime. It is interesting to note that at smaller graph sizes there is little variation in runtime, as it is likely that Spark has a fixed initialisation time (JVM initialisation time, YARN scheduling delay and data distribution) for a job of any dataset size.

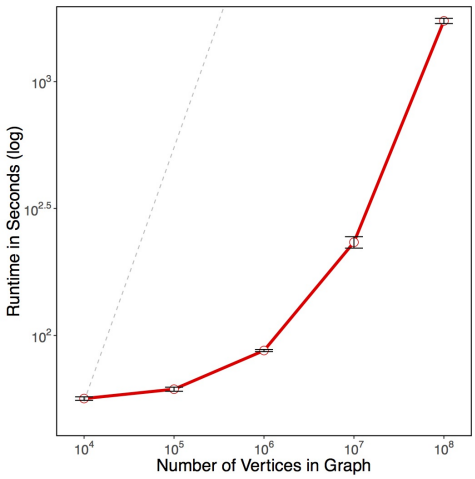


Figure 5. Runtime Across Number Of Vertices In A Forest Fire Graph

Figure 6 shows how GFP-X responds to increases in the number of vertices within an Erdős-Rényi graph. Again it can be seen that the GFP-X approach scales approximately sub-linearly to increases in dataset size. Certainly below 10^7 vertices, the increase in runtime can be considered sub-linear. However the increase from 10^7 to 10^8 requires moderately more than linear time perhaps owning to the random nature of the topologies of Erdős-Rényi graphs not parallelising well. However below 10^8 vertices, the profile of the runtime performance of the Erdős-Rényi run is very similar to the profile of the runtime for the Forest Fire graphs. This suggests that the runtime of the GFP-X is largely independent of the topological structure of the graph being fingerprinted, a desirable quality for a graph mining algorithm.

F. Discussion

The GFP-C approach outperforms the current state of the art feature based extraction methods, displaying excellent runtime and can scale to previously unmanageable graph sizes. The GFP-C approach is sensitive to detecting small variations in graph topology and overall graph size. Due

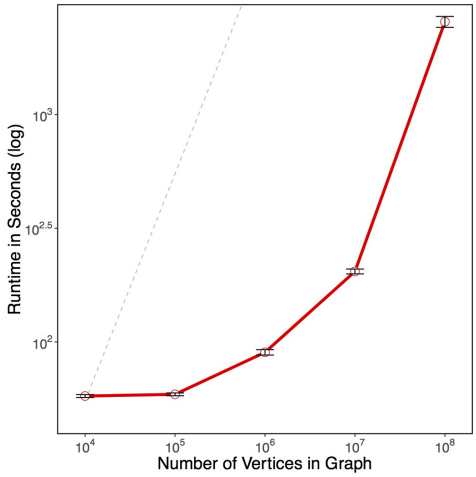


Figure 6. Runtime Across Number Vertices In A Erdős-Rényi Graph

to the nature of the features extracted, the GFP-X approach requires no labels with the graph datasets. However, perhaps the most promising result to arise is the sub-linear runtime of the approach when increasing dataset size up-to 10^8 vertices on a modest 4 node Spark cluster. This has the potential to improve machine learning based approaches to temporal graph analysis as there is now an efficient way to validate models against empirical data.

The GFP-X approach is effectively able to take the high-dimensional complexity inherent in graph datasets, and reduce it to a single fingerprint vector. There are numerous other applications, outside of similarity measures, that could massively benefit from a compact representation of a graph. The application of modern deep learning techniques upon graph datasets is largely unexplored [32], and the fingerprint vector could be a key aspect in unlocking the use of an extended range of these techniques. An example could be the classification of unknown datasets by comparing their fingerprint vectors to labeled ground-truth datasets. For example, do datasets from a certain domain have fingerprints unique enough that they can be used to classify the domain?

VIII. Conclusion

In this paper, the Graph FingerPrint Comparison approach for assessing the similarity of two unlabelled graphs, based upon their macro and micro features, has been presented. The GFP-X fingerprint generation exploits Apache Spark and GraphX to extract powerful, neighbourhood based, features from a graph in parallel. When comparing, the GFP-C approach is shown to be sensitive to small variations in graph topology, graph size and function without the requirement of labelled datasets whilst also scaling nearly sub linearly with dataset size across a Spark cluster. Thus the GFP-C approach completes all of the goals established for it in section III-A. The approach demonstrates promising results and the concept of a compact but accurate representation of a graph has numerous potential additional applications

within machine learning.

There is large scope for future research based upon the work presented in this paper. We would like to investigate how the approach scales to even larger graphs and as more compute nodes are added to the Spark cluster. Apache Spark is developing a new graph API called GraphFrames, based upon DataFrames, which promises even greater performance, we would like to port GFP-C to this and compare performance. In addition to this, the application of the extracted graph fingerprints to other use cases within network science will be explored. For example, could a graph's fingerprint be used to study the temporal evolution of a graph? We foresee the applications of studying a graph's fingerprint will be numerous.

ACKNOWLEDGMENT

The authors would like to thank the Engineering and Physical Sciences Research Council (EPSRC) for funding.

REFERENCES

[1] L. Akoglu, H. Tong, and D. Koutra, "Graph based anomaly detection and description: a survey," *Data Mining and Knowledge Discovery*, 2015.

[2] S. Bonner, A. S. McGough, I. Kureshi, J. Brennan, G. Theodoropoulos, L. Moss, D. Corsar, and G. Antoniou, "Data quality assessment and anomaly detection via map/reduce and linked data: A case study in the medical domain," in *2015 IEEE International Conference on Big Data (Big Data)*. IEEE, 2015.

[3] L. A. Zager and G. C. Verghese, "Graph similarity scoring and matching," *Applied Mathematics Letters*, 2008.

[4] N. Przulj, "Biological network comparison using graphlet degree distribution," *Bioinformatics*, 2007.

[5] C. Aggarwal and K. Subbian, "Evolutionary network analysis: A survey," *ACM Computing Surveys (CSUR)*.

[6] M. Berlingerio, D. Koutra, T. Eliassi-Rad, and C. Faloutsos, "NetSimile: A scalable approach to size-independent network similarity," 2012.

[7] D. Koutra, A. Parikh, A. Ramdas, and J. Xiang, "Algorithms for Graph Similarity and Subgraph Matching," Tech. Rep., 2011.

[8] P. Papadimitriou, A. Dasdan, and H. Garcia-Molina, "Web graph similarity for anomaly detection," *Journal of Internet Services and Applications*, 2010.

[9] G. Malewicz, M. Austern, and A. Bik, "Pregel: a system for large-scale graph processing," *Proceedings of the 2010 ACM SIGMOD International Conference on Management of data*, 2010.

[10] N. Doekemeijer and A. L. Varbanescu, "A Survey of Parallel Graph Processing Frameworks," 2014.

[11] S. Bonner, J. Brennan, I. Kureshi, G. Theodoropoulos, and A. S. McGough, "Efficient Comparison of Massive Graphs Through The Use Of Graph Fingerprints," in *Twelfth Workshop on Mining and Learning with Graphs (MLG) Workshop at KDD'16*. ACM, 2016.

[12] M. S. M. Vijay, M. Vijesh, S. Iyengar, S. R. Nayak, N. Shenoy, and R. Sundaram, "Prediction of arrival of nodes in a scale free network," *Proceedings of the 2012 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining, ASONAM 2012*, 2012.

[13] M. Roy, S. Schmid, and G. Tredan, "Modeling and measuring graph similarity," in *Proceedings of the 10th ACM international workshop on Foundations of mobile computing - FOMC '14*, New York, New York, USA, 2014.

[14] G. Li, M. Semerci, B. Yener, and M. J. Zaki, "Effective graph classification based on topological and label attributes," *Statistical Analysis and Data Mining*, 2012.

[15] D. Koutra, J. T. Vogelstein, and C. Faloutsos, "DELTACON: A Principled Massive-Graph Similarity Function," *ACM Transactions on Knowledge Discovery from Data*, 2016.

[16] G. Kollias, M. Sathe, O. Schenk, and A. Grama, "Fast parallel algorithms for graph similarity and matching," *Journal of Parallel and Distributed Computing*, 2014.

[17] G. N. Lance and W. T. Williams, "Mixed-data classificatory programs i - agglomerative systems." *Australian Computer Journal*, 1967.

[18] P. Bonacich, "Some unique properties of eigenvector centrality," *Social Networks*, 2007.

[19] L. Page, S. Brin, R. Motwani, and T. Winograd, "The PageRank citation ranking:bringing order to the web." 1998.

[20] M. Han, K. Daudjee, K. Ammar, M. T. Ozsu, X. Wang, and T. Jin, "An Experimental Comparison of Pregel-like Graph Processing Systems," *Vldbe*, 2014.

[21] D. J. Watts and S. H. Strogatz, "Collective dynamics of 'small-world' networks." *Nature*, 1998.

[22] M. Zaharia, M. Chowdhury, M. J. Franklin, S. Shenker, and I. Stoica, "Spark : Cluster Computing with Working Sets," *HotCloud'10 Proceedings of the 2nd USENIX conference on Hot topics in cloud computing*, 2010.

[23] M. Armbrust, R. Xin, and M. Zaharia, "Spark SQL: Relational Data Processing in Spark," *Sigmod'15*, 2015.

[24] R. S. Xin, J. E. Gonzalez, M. J. Franklin, and I. Stoica, "GraphX: A Resilient Distributed Graph System on Spark," *First International Workshop on Graph Data Management Experiences and Systems*, 2013.

[25] R. R. McCune, T. Weninger, and G. Madey, "Thinking Like a Vertex," *ACM Computing Surveys*, 2015.

[26] K. T. Bartusiak R., "Sparklinggraph: large scale, distributed graph processing made easy," 2016, manuscript in preparation.

[27] T. de Paula Peixoto, "graph-tool: An efficient python module for manipulation and statistical analysis of graphs. ," 2016. [Online]. Available: https://graph-tool.skewed.de/

[28] J. Leskovec, J. Kleinberg, and C. Faloutsos, "Graph Evolution: Densification and Shrinking Diameters," *ACM Transactions on Knowledge Discovery from Data*, 2007.

[29] P. Erdős and A. Rényi, "On random graphs I," *Publicationes Mathematicae*, 1959.

[30] J. Leskovec and R. Sosič, "Snap: A general-purpose network analysis and graph-mining library," *ACM Transactions on Intelligent Systems and Technology (TIST)*, 2016.

[31] J. Leskovec and A. Krevl, "SNAP Datasets: Stanford large network dataset collection," http://snap.stanford.edu/data, 2014.

[32] M. Henaff, J. Bruna, and Y. LeCun, "Deep Convolutional Networks on Graph-Structured Data," *arXiv*, 2015.